Preface

THE ADVENTURE SYSTEM contains programs for creating/editing adventure data bases and running the data bases. This package requires a 48K disk system.

ADVLIST (the original adventure creator/editor) was written by Allan Moluf for his own use. This version was written in BASIC and worked quite well. Bruce Hansen enhanced ADVLIST with new commands. Since some ADVLIST commands could take close to three minutes to complete their tasks, the slower routines were written in assembly language by Bruce Hansen. The program was consequently renamed ADVEDIT.

The ADVENTUR/CMD driver program written by Scott Adams could not be sold with this package as this would be a copyright violation. Moreover, this program is not readily accessible from the newer adventure diskettes which are protected. However, this driver program is required to run the data bases created/edited by ADVEDIT. To alleviate this problem, the ADV/CMD program was written by Bruce Hansen. This program functions exactly as Scott Adams' Adventure driver program.

A disk to tape utility program is available from the dealer for an extra charge. The ADVTAPE/CMD program will read in a disk adventure data base and write out a tape version of the adventure. The tape will contain the adventure driver written by Bruce Hansen and the data from the disk data base file. This allows a 16K computer to run adventures created with ADVEDIT. Since the tape contains the adventure driver written by Hansen, these tape versions can not be sold without written permission from the author. For more information on selling your adventures, see Appendix B of this manual.

Since ADVEDIT requires the data bases to not be on protected diskettes, the ADVCOPY/CMD utility was written. This program will read the data base from a Scott Adams' protected diskette and put it on an unprotected one. This program is also available from your dealer for an extra charge.

A great deal of work has been put into this manual. It is intended to instruct the user on everything he needs to know to write and edit adventures. You supply the ideas.

The saying "if all else fails, read the instructions" should read "all will fail without reading the instructions" when applied to the ADVEDIT program. The ADVENTURE program written by Scott Adams has over 60 commands and conditions, and many subtle rules. Don't worry about learning all of the commands however, just refer to this manual as needed.

Later readings will make finer details easier to pick up.

If you have any suggestions for improvements to ADVEDIT, please send them to:

Bruce G. Hansen
220 Iris Street
Lansing, MI 48917
(517) 323-2260

Any adventures you write are your property. You may market them on your own if you so desire. However, these adventures require an adventure driver to run them. The two available adventure driver programs are copyrighted by their respective authors and may not be distributed with any adventures you sell unless written permission is given by the author. The primary dealer of this package, THE ALTERNATE SOURCE, will market all acceptable adventures and pay a substantial royalty. See Appendix B for more details. TAS, and only TAS, has been authorized to use the adventure driver program written by Bruce Hansen. Perspective authors should contact THE ALTERNATE SOURCE for more information. They may be contacted by writing or calling to the following address/phone number:

THE ALTERNATE SOURCE
1806 Ada Street
Lansing, MI 48910

Introduction

ADVEDIT is an editing program for adventures. With it, most adventure data bases can be read in and viewed or modified. Even a Scott Adams' ADVENTURE can be read in and solved. This manual is divided into chapters, each containing a different section of ADVENTURE and using the ADVEDIT program. A summary of the chapters and appendices is given below:

Chapter 1	Overview of ADVEDIT and adventures in general. This chapter describes what ADVEDIT does and the basics of adventure.
Chapter 2	Description of the ADVENTURE data base structure. This chapter will describe in detail what the different conditions and commands of ADVENTURE are. This is the most important chapter to know before trying to write your own adventures.
Chapter 3	ADVENTURE program instructions. This chapter describes how adventures must be entered so they will work properly with Scott Adams' ADVENTURE program or Bruce Hansen's ADV/CMD program.
Chapter 4	Operating Instructions. This chapter contains the instructions for ADVEDIT. A suggested procedure is also given to assist you in entering an adventure.
Chapter 5	Sample ADVENTURE. This chapter will describe a short adventure written to show how to use most of the commands and conditions.
Chapter 6	ADVENTURE solving techniques. This chapter will describe how to solve any adventure readable by the ADVEDIT program.
Appendix A	contains an abbreviated ADVEDIT command summary. This will be highly useful when writing adventures once you are familiar with the commands and conditions.
Appendix B	tells how to submit your adventures to THE ALTERNATE SOURCE for marketing purposes.

Chapter 1

Overview of ADVEDIT and Adventures in general

ADVEDIT has only one purpose: To edit and create adventure data bases. All user created adventures must be run with either Scott Adams' ADVENTURE program (version 8.2) or Bruce Hansen's ADV program. ADVENTUR/CMD is the file name for the main or driver program. Adventure data bases are saved on disk as ADVENT/Dx, where x is a number from 0-9 or a letter A-Z. Adventures X and Y are on your master diskette as ADVENT/DX and ADVENT/DY. Adventure "X" is a "Miner's adventure" and adventure "Y" is a "Burglar's adventure".

Recently Scott Adams started selling all of his adventures on "protected" diskettes. The adventure data bases on these diskettes are unusable by ADVEDIT. However, a program is available from your dealer which will transfer the data base from a protected diskette to an un-protected one. This program is called "ADVCOPY". The adventure driver program on these protected diskettes can not be copied to an unprotected diskette thus raising the need for ADV/CMD.

ADVENTURE 8.2 (by Adams') has recently been superceded by version 8.3. The newest version is included on all Adventure 0-12 diskettes. ADVEDIT supports all commands of Adventure 8.2 and 8.3, however, as new versions are released, new commands may be added. This may make ADVEDIT and ADV/CMD incompatible with the adventure data bases used by this new version. ADVEDIT and ADV/CMD will be maintained so they work with new versions of ADVENTURE. At this time, ADVEDIT and ADV/CMD will work with Scott Adams' Adventures 0-12.

The concept of adventure is very simple. The basic idea is that certain commands are executed when certain conditions are met. The only problem, as all adventurers have found, is getting the conditions to be met. The trick to writing good adventures (like Scott's) is making the conditions subtle, but logical. The purchase of Scott's adventures is highly recommended. The techniques in his data bases are excellent teaching tools along with being fun to play.

Chapter 2

Adventure data base format

The data base is the actual adventure. By changing this data base different adventures may be obtained.

The data base consists of the following sections:

1) HEADER information. The header contains the number of actions, vocabulary entries, rooms, messages, objects and some other variables.

2) ACTION entries. The action entries contain the known player inputs (verb, noun), conditions and commands. Also contained are automatic actions. The automatic actions serve mainly for bookkeeping.

3) VOCABULARY entries. These two lists (verbs and nouns) contain all of the words the player may use in this particular adventure.

4) MESSAGE text. These are the messages used by the adventure and are controlled by the actions.

5) ROOM description. This is a list of directions to other rooms along with a text room description.

6) OBJECT description and starting locations. The description of the object determines if it is a treasure, an object which may be carried and dropped or something else. The starting location tells in which room the object starts in, if it is being carried at the start of the adventure or if it starts in the storeroom.

7) ACTION TITLES. These are optional text descriptions of the actions. They are ignored by the ADVENTURE driver program but serve as remarks to the actions when using the ADVEDIT program.

8) TRAILER information. This contains the version number, adventure number and a checksum.

HEADER information

The header contains the following information:

1) The number of bytes required to hold all of the text descriptions such as verbs, nouns, messages, room descriptions and object descriptions. This number includes a fixed number of bytes for each verb and noun. This fixed number is the word length of this

adventure plus one. It includes one more than the number of characters between quotes in the messages and room and object descriptions. The number of bytes specified may be larger than necessary, but must not be smaller or the ADVENTURE driver program will tell how much too small and quit.

2) The highest numbered object in this particular adventure. The objects are numbered starting at zero, so the number of objects is one plus this number.

3) The highest numbered action in this particular adventure. Actions are numbered starting at zero, so the number of actions is one plus this number.

4) The highest numbered vocabulary word in this adventure. This applies to both verbs and nouns, being the larger value if they are different. Vocabulary words are numbered starting at zero, so the total number of verbs and total number of nouns is one plus this number.

5) The highest numbered room in this adventure. Rooms are numbered starting at zero, but room zero is reserved as a storeroom so the total number of rooms in which the player may enter is this value.

6) The maximum number of objects which may be carried. Under certain conditions the actions can cause more than this number to be carried. The player will not be able to pick up anything unless the number of objects currently being carried is less than this number.

7) The starting room number for this adventure.

8) The number of treasures in this adventure. When the SCORE command is issued this number is divided by the number of treasures in the treasure room to give the percent score.

9) The word length used by this adventure. This number affects the nouns and verbs. When the adventure data base is read in by the ADVENTURE driver program, all nouns and verbs are either truncated or padded to this length plus one. This value is the minimum length of verbs and nouns the player may input.

10) The time limit. This may be used in some games to control how long the artificial light will last. If there is no artificial light, it may control the number of turns in this adventure. If the artificial light is re-filled, this value is put back in the time limit. The limit is 32767.

11) The highest numbered message. Messages are numbered

from zero so this value is the number of messages plus one.

12) The treasure room number. When treasures are in this room they are considered collected. When the SCORE command is issued they are summed and divided into the number of treasures for the percentage score.

ACTION entries

These are the heart of the adventure. Some are player input and others are automatic operation actions. The entries are stored as eight numbers. The first determines when the action is to be evaluated. The next five are conditions to be met or parameters for the commands. The last two bytes specify what commands are to be performed if all of the conditions were met.

The first number is (150*verb + noun). If the verb is zero, this is an automatic action and the noun number (1-100) is the probability of this action being evaluated. If the verb is not zero, it must match the verb in the player's input and the noun must match the noun in the player's input for the action to be considered. If the noun is zero, it matches any possible noun in the player's input.

When player input actions are being evaluated, the action entries are scanned in numeric order. When the verb and noun of an action entry match the player's input, the conditions are evaluated. If all of the conditions were true, then the commands of this entry are performed. When a "true" match is made, no further player input actions are evaluated on this pass. However, if a match is made and all of the conditions were not true, then the scanning procedure continues until either a "true" match is made or all of the actions are evaluated. If a match was found but the conditions were not true, then the message "I can't do that . . . yet!" is displayed. If no match was found, then the message "I must be stupid but I don't understand what you mean" is display.

However, when automatic actions are being evaluated, they are all scanned regardless of how many are true or false.

If the action is to be considered, the five conditions are evaluated. If any conditions fail, the commands in the action are not performed. The conditions are (20 * number + condition). The condition codes and their meanings are as follows:

PAR This condition always passes. The number included with PAR (i.e. PAR 20) may be used by the commands in this entry. See the list of commands for uses of parameters.

HAS	The condition passes if the player is carrying the numbered object (i.e. HAS 15). It fails if the object is either in the same room as the player or in any other room.
IN/W	The condition passes if the player is in the same room as the numbered object. It fails if the player is either holding the object or the object is in any other room.
AVL	The condition passes if the numbered object is available because the player is either carrying the object or in the same room as the object. It fails if the object is in any other room.
IN	The condition passes if the player is in the numbered room (i.e. IN 5). It fails if the player is in any other room.
-IN/W	The condition passes if the numbered object is held by the player or if the object is in any other room. It fails if the object is in the same room as the player.
-HAVE	The condition passes if the player is not carrying the numbered object. It fails if the player is carrying the object.
-IN	The condition passes if the player is not in the numbered room. The condition fails if the player is in any other room.
BIT	The condition passes if the numbered bit flag is set. It fails if the flag is cleared. See the description of bit flags later on for more information.
-BIT	The condition passes if the numbered bit flag is cleared. It fails if the flag is set. See the description of bit flags later on for more information.
ANY	The condition passes if the player is carrying any objects at all. It fails if the player is not carrying any objects. The parameter entered (i.e. ANY 50) has no affect on this condition.
-ANY	The condition passes if the player is not carrying any objects. It fails if the player is carrying any objects at all.
-AVL	The condition passes if the numbered object is in

	any other room. It fails if the object is available either because it is being carried or it is in the same room as the player.
-RMO	The condition passes if the numbered object is not in room zero. Room zero is reserved as a storeroom. The condition fails if the object is in room zero.
RMO	The condition passes if the numbered object is in room zero. The condition fails if the object is in any room other than room zero.
CT<=	The condition passes if the counter is less than or equal to the number. It fails if the counter is greater than the number. See the description of the counters later on for more information.
CT>	The condition passes if the counter is greater than the number. It fails if the counter is less than or equal to the number.
ORIG	The condition passes if the numbered object is in the same room it started in. It fails if the object is in any other room or is being carried.
-ORIG	The condition passes if the numbered object is in any room other than its starting room or is being carried. It fails if the object is in the same room it started in.
CT=	The condition passes if the counter is equal to the number. It fails if the counter is not equal to the number.

BIT FLAGS

There are thirty-two bit flags available to the user. They are numbered 0 to 31. When the adventure is started, they are all cleared. There are commands to set and clear them as well as conditions to test their values. Two bit flags are reserved by ADVENTURE 8.2 and ADV:

15) If this bit flag is set it is dark outside. The room will be in darkness unless the artificial light source is available. The artificial light source is discussed in the OBJECT section of this chapter. There are two commands (DAY and NIGHT) to clear and set this bit flag.

16) When this flag is set the artificial light source has run out. The "FILL" command will clear this flag and set the time limit to its original value.

COUNTERS

The counters are values which may be incremented, decremented, assigned values by commands as well as be tested against a number for numeric conditions. There are alternate counters which may be switched with the CT counter in order to operate on other numbers. When the adventure is started, CT is not assigned any particular value. See Chapter 5 for more details on how counters may be used in adventures.

ALTERNATE ROOM REGISTERS

The value of the current room may be saved and restored by exchanging it with an alternate room register. The saved room value may be restored by performing another exchange with the same alternate room register.

The seventh and eighth bytes of the action entry are the four command codes. The seventh number is (150*command 1 + command 2) and the eighth number is (150*command 3 + command 4).

These four commands may use one or more parameters found in the condition line of the same action entry. For example, if the first parameter found in the conditions was a 10 (PAR 10) and the first command which used a parameter in the commands was a GOTOY command, the player would move to room 10 (GOTO 10).

If a command uses one parameter, its value is represented by "Par #1" in the following command description. If the command uses two parameters the first is represented by "Par #1" and the second by "Par #2." The parameters used by any command are skipped by later commands if they also use parameters. For example, if the conditions held three parameters: PAR 3, PAR 15, PAR 26 in that order, the first command that used a parameter would use the 3, the second command would use the 15 and the third would use the 26. Too many parameters in the conditions has no effect. But not having as many parameters in the conditions as the number expected by the commands will produce strange results. For example, specifying no parameters in the conditions and having a "GETX" command in the commands.

These are the possible command codes in ADVENTURE 8.3:

0	No command or message.
1-51	Display message numbers 1-51.

52	GETX	Pick up Par #1 object unless the player is already carrying the maximum number or limit. The object may be in the current room or in any other room.
53	DROPX	Drop the Par #1 object in the same room as the player. The object may be carried or in another room.
54	GOTOY	Move the player to the Par #1 room. This command should be followed by a DSPRM command. Also, this may need to be followed by a DAY/NIGHT command depending on the light status of the room.
55	X-RMO	This command moves the Par #1 object to room zero.
56	NIGHT	This command sets the light/darkness bit flag (15). The room will be dark if the artificial light source is not available. This command should be followed by a DSPRM command.
57	DAY	Clear the light/darkness bit flag (15). This should also be followed by a DSPRM command.
58	SETZ	Set the Par #1 bit flag.
59	X->RMO	This command is a repeat of command 55.
60	CLRZ	This clears the Par #1 bit flag.
61	DEAD	This clears the light/darkness flag (makes it light), moves the player to the last room and tells him he is dead.
62	X->Y	Move the Par #1 object to Par #2 room. This command will automatically display the room if the Par #1 object either entered or exited the current room.
63	FINI	Indicate to the player that the game is over and inquire if he wants to play again.
64	DSPRM	Display the current room. This checks the light/darkness flag and if the artificial light source is present. If it is light, the room description, visible objects and obvious exits are displayed. If it is dark, nothing is displayed (it is too dark to see) unless the artificial light source is present.
65	SCORE	Tells the player how many treasures are in the

treasure room and what percentage the total is. If one hundred percent is stored, then the winning message is displayed and the player is given the option of playing again.

66	INV	Tells the player what objects are being carried.
67	SETO	This sets the zero bit flag. It may be useful since no parameter from the conditions is necessary.
68	CLRO	Clears the zero bit flag. It may be useful since no parameter from the conditions is necessary.
69	FILL	Re-fills the artificial light source and clears the bit flag 16 (indicator of light source status). This also picks up the artificial light source. This command should immediately be followed by a X->RMO command where Par #1 is the unlighted artificial light source (they are two different objects).
70	CLS	This command did a clear screen in the BASIC version of ADVENTURE and does nothing in the machine language version.
71	SAVE	Saves the game to disk or tape depending on which version is being used. It writes some user variables such as the current room, current locations of all objects, status of all bit flags, current values of all alternate room registers and the current values of all counters.
72	EXX,X	Exchange the room location of the Par #1 object with the room location of the Par #2 object. A DSPRM is automatically performed if either Par #1 or Par #2 objects were in the current room.
73	CONT	This command sets a flag to allow more than four commands to be performed. When all commands in this action entry have been performed, the conditions of all subsequent action entries with a zero verb and noun (up to the first non-zero verb and noun) will be evaluated. The checking procedure continues regardless if the entry being checked is true or false. For example, consider the following actions:

```
LIGHT TORCH   HAS 12   PAR 9   PAR 12   PAR 0    PAR 0
      EXX,X   MSG5   CONT  -
```

```
AUTO  0        PAR 1    PAR 0    PAR 0    PAR 0    PAR 0
     EXM,CT   CT-1       -        -
AUTO  0        CT= 0    PAR 9    PAR 12   PAR 0    PAR 0
     EXX,X    MSG6       -        -
AUTO  0        PAR 1    PAR 0    PAR 0    PAR 0    PAR 0
     EXM,CT    -         -        -
SHOOT GUN      HAS 23   IN/W 2   PAR 2    PAR 4    PAR 0
     EXX,X    MSG8       -        -
```

If the conditions of the action with the verb-noun of "LIGHT TORCH" are found to be true, then its commands are executed. One of the commands is a "CONT". This means that all "AUTO 0" verb-noun actions following "LIGHT TORCH" will be considered. In this case there are three of them. All three are considered even if none of them are true or false. For example, the third one is considered even if the second one was true.

74	AGETX	Always get Par #1 object even if the carry limit is overflowed.
75	BYX->X	Put the Par #1 object in the same room as the Par #2 object. If the Par #2 object is being carried this will pick up the Par #1 object also, regardless of the carry limit. If this command changes any objects in the current room a DSPRM command is automatically executed.
76	DSPRM	This is a copy of command 64.
77	CT-1	Subtract one from the counter value.
78	DSPCT	This displays the value of the counter. No carriage return is printed after the value.
79	CT<-N	The sets the counter equal to the Par #1 value.
80	EXRM0	This exchanges the current room with the room number held in alternate room register zero. This may be used to save a player's current room for return to it later on. This command should be followed by a GOTOY command if the alternate room register zero had not been set.
81	EXM,CT	Exchange the value of the counter and the value of the Par #1 alternate counter. There are eight counters numbered 0 to 7. When the adventure starts these are not set to any particular value so initialization automatic action entries should set them. Also, the time limit may be accessed by exchanging with alternate counter eight (8).

82	CT+N	Add the Par #1 value to the counter.
83	CT-N	Subtract the Par #1 value from the counter.
84	SAYW	This displays the noun (second word) input by the player.
85	SAYWCR	This displays the noun (second word) input by the player followed by a carriage return.
86	SAYCR	Starts a new line on the display.
87	EXC,CR	Exchange the value of the current room with the Par #1 alternate room register. This may be used to remember more than one room. There are six alternate room registers numbered 0 to 5.
88	DELAY	This command pauses for about 1 second before going on to the next command.
89-101		These commands are undefined by version 8.3 of ADVENTURE but may be used in future ADVENTURE releases.
102-149		Display messages 52-99.

Note that action commands 89-101 are not used. Scott Adams may use these in future updates of the adventure driver. However, they may also be defined by THE ADVENTURE SYSTEM. If you have any suggestions for added commands, send them to the author for consideration.

The automatic action entries have a variety of uses. All of them are considered before a player input. Such things as falling asleep, checking for day/night or any other tasks that must be performed without player input are candidates for automatic action entries. Chapter 5 contains a more detailed description of automatic actions.

VOCABULARY entries

Each vocabulary entry consists of a verb string and a noun string. Synonyms are handled by beginning the word with an asterisk, which are then treated the same as the first previous word with out an asterisk. Some of the vocabulary entries are predefined by ADVENTURE and SHOULD NOT be changed. These predefined verbs and nouns are listed below:

Verbs

0	AUTO	This is not entered by the player while playing

the adventure. It signifies the auto action entries which are all evaluated before a valid player input.

1	GO	This is a special case for the direction nouns 1-6.
10	GET	This is used to pick up objects if there is no action entry that applies and the noun matches the name enclosed in slashes in an object name in the current room. See the OBJECT section of this chapter for more information on the object name.
18	DROP	This is used to drop objects if there is no action entry that applies and the noun matches the name enclosed in slashes in an object name being carried.

Nouns

0	ANY	This is not entered by the player while playing the adventure. It denotes the action entries which can match any noun (or no noun).
1	NORTH	This is reserved for the first room direction entry with verb 1.
2	SOUTH	This is reserved for the second room direction entry with verb 1.
3	EAST	This is reserved for the third room direction entry with verb 1.
4	WEST	This is reserved for the fourth room direction entry with verb 1.
5	UP	This is reserved for the fifth room direction entry with verb 1.
6	DOWN	This is reserved for the sixth room direction entry with verb 1.

ROOM entries

The room entries consist of the number of the adjacent room in the six reserved directions N, S, E, W, U and D plus a room description string. If the adjacent room number is zero, there is "no obvious exit" in that direction. If the adjacent room number in the N direction is 5, then going NORTH will put the player in room 5.

If the text description of the room does not begin with an asterisk, the ADVENTURE program will precede the string with "You're in a"; otherwise, it will just display the description minus the asterisk. To get quotes (") within the text description type a SHIFTed @ in place of the quotes. A SHIFTed @ sign will look like a regular @ sign unless an upper/lower case mod has been installed and some video driver program is being used. The ADVENTURE program will automatically change these to quotes.

Room zero is reserved as a storeroom for objects currently not in any room. The player can not get to room zero by using one of the reserved directions. Actions usually do not permit the player to enter this room.

The last room is reserved for some sort of limbo state should the player die. This is where the player is sent with a DEAD command. It may or may not contain exits back to the other rooms.

MESSAGE entries

The messages consists of a string of characters for each message to be displayed by any of the action entries. Entry 0 should always be left as a null string. To get quotes to appear within a message simply type a SHIFTed @ in place of the quote. ADVENTURE will change this to a quote.

OBJECT entries

The object entries consist of a text description of the object along with its starting room number. Room zero is used for objects not found yet. A minus one (-1) is used for the starting room when the player is carrying that object at the beginning of the game.

The object descriptions should begin with an asterisk if that particular object is a treasure. Also, if the object is to be picked up and dropped, the word to use for it is enclosed in slashes at the end of the description. The word between the slashes must be the same length or smaller than the word length of the adventure. If the verb is 10 (GET) or 18 (DROP) and no other action applies, the adventure program will automatically pick up or drop the object if the player's inputted noun is the same as the object name. The name of the object must be a noun in the list of vocabulary entries for this pick up and drop feature to work. The object name must also be a primary noun, not a synonym.

An example of a treasure that can be picked up is:

FIRESTONE (cold now)/FIR/

which can be picked up by the word "FIR". Before the firestone is cooled, the treasure was in the storeroom and the following object was in the room:

glowing *FIRESTONE*

Because this object does not begin with an asterisk it is not recognized as a treasure. Also, it can not be picked up since it has no name between slashes. The action that cools the firestone exchanges the locations of these two objects.

Object number nine (9) is reserved as the artificial light source in its lighted state. The ADVENTURE program checks to see if object 9 is available when a room is in darkness (NIGHT). Also, the FILL command GETs object 9 when the light is recharged. Examples of object 9 are a lit flashlight and a lit lamp.

ACTION TITLES

The action titles are labels for the action entries. They aid in commenting the actions and ease in editing the adventure. The ADVENTURE program discards the titles when an adventure is read in because they are only used by an adventure editor program. These descriptions SHOULD NOT contain quote marks.

TRAILER

The trailer information contains the version number, the adventure number and a security checksum. If the version number was 415 it will be displayed as "4.15". The adventure number is simply the number identifying the adventure (0-9, A-Z). The security checksum is (2 * #actions + #objects + version). If the checksum computed by the ADVENTURE program does not equal the one in the adventure file the ADVENTURE program will hang up.

Chapter 3

ADVENTURE Instructions

This chapter will give some rules governing the entering of a user adventure. These are rules pertaining to Scott Adams' ADVENTURE program and Bruce Hansen's ADV program, not ADVEDIT. Rules are given in sections such as Action entries, etc.

Rules for the Action entries are as follows:

1) All of the automatic actions must proceed player input actions. If the auto-actions are not placed first they will be ignored.

2) If the action entry uses commands which require parameters, there must be parameters in the condition line. If not, strange messages and objects will appear.

Rules for the vocabulary:

1) The predefined verbs and nouns (NORTH, GET, etc.) must remain in their preset positions. Failure to do so will cause difficulty in moving from room to room and/or carrying objects.

2) This is not a rule, but a strong suggestion. When keying in verbs and nouns with ADVEDIT, use only the word length specified in the header. This will make unintentional duplicate words easy to find. Duplicate nouns and verbs can be a big problem. For example, suppose noun 10 was SHED and noun 11 was SHELF. If the word length was three, SHED and SHELF would appear to be the same noun, SHE. This is where the problem occurs. If an action entry refers to SHELF (i.e. EXAMINE SHELF), the ADVENTURE driver program starts scanning the list of nouns for a match. The first occurrence of SHE is in SHED and since the noun number ADVENTURE is looking for (11) is not the same as the one it found (10), the action does not work. By limiting the length of inputted nouns and verbs to the word length, this problem will never crop up.

3) If a word is to be a synonym, it should be preceded by an asterisk and placed after the primary noun or verb. For example, if GO is the verb and RUN, WALK and ENTER are to be synonyms, the list of verbs would read GO, *RUN, *WALK and *ENTER.

4) Nouns and verbs must not contain embedded quotes.

Room rules are as follows:

1) No quotes can be used in the room description. Quote marks are used as delimiters for the room description, thus using them within a description will cause problems when trying to read the adventure data base. If quote marks are desired, a SHIFTed @ sign (a character 60 hex) should be used in their place. The ADVENTURE program will change these to quotes (the SHIFTed @ sign will look like a normal @ sign unless a software video driver is being used).

2) The header contains the number of rooms, the last being used to send the player to after a DEAD command. The last room should be some sort of limbo state or something similar.

3) Each room has six values associated with it. These are the room numbers which are entered on a direction command (i. e. GO NORTH). These values should be legal room numbers (ADVEDIT won't let a bad number be entered). A zero is used if no exit is possible in that direction.

The rules for Messages are as follows:

1) No quotes can be embedded in the messages. Quote marks are used as delimiters for the messages, thus using quotes within a message will cause problems with a disk read. To get quote marks within a message, type SHIFTed @ signs in their place. The ADVENTURE program will change these to quote marks.

2) Message 0 should be null.

Object rules:

1) The starting room for an object should be a valid room number. Objects not found yet or not used at the beginning of the adventure should be in room zero (the storeroom). Objects the player is carrying at the start of the adventure should have a starting room of minus one.

2) The object description should not contain any embedded quote marks. Quote marks are used as description delimiters while stored on disk and embedded quote marks will mess up the disk file. Use a SHIFTed @ sign in place of quotes. The ADVENTURE program will change these to quotes after they are read in.

3) An object name is placed between slashes at the end of the object description if it is to be carried and dropped. The name should be the same length or less than the word length of the adventure. The object name must be a primary noun in the vocabulary list, not a synonym. If it is a synonym the pick up and drop feature will not work for that object.

There are a few differences in the ADVENTURE drivers "ADV" and "ADVENTUR". The biggest difference is the disk input/output of each.

"ADVENTUR" limits the name of the adventure data base being read in to one character. For example, legal adventure names are:

```
ADVENT/DA
ADVENT/D7
```

"ADV" limits the name of the adventure data base to two characters. For example, legal adventure names are:

```
ADVENT/DA
ADVENT/D7
ADVENT/DAA
ADVENT/DOA
```

The other difference is the file structure of a game saved in progress. These saved games are not compatible with each other (a game saved by "ADV" can not be read in by "ADVENTUR").

The file names written out are also different. The main part of the name is the same - the player's name input at the beginning of the adventure, but the file name extensions are different.

With "ADVENTUR", the extension is "/Sx" where "x" is the adventure number being played.

With "ADV", the extension is "/Sxn" where "x" is the first character of the adventure number (the second one is ignored if the adventure data base had a two letter name like ADVENT/DOA) and "n" is the game version written out. "ADV" allows up to ten different files to be written from the same adventure. This allows the progress of an adventure to be saved out at different points.

Other than these simple differences, the two programs are nearly identical.

Chapter 4

ADVEDIT Instructions

This chapter contains the instructions for the ADVEDIT program. Each option will be covered in detail.

Program start up procedure:

Put the ADVEDIT diskette in drive 0 and turn on the computer. From the DOS command mode type:

```
XREF
```

Under no condition should you just enter "LOAD XREF/CMD". The program contains a relocating loader which moves it up to high memory. Just "LOADing" the program will not relocate it to high memory. Also, never load the "XREF" program twice without resetting the computer. The program will still work except the second load will simply set lower in memory than the first one thus wasting memory.

If any high memory drivers are to be used, they must be loaded before the "XREF" program. They must also protect themselves by setting the high memory pointer (4049H for Model I, 4411H for Model III) to some value below them. Failure to do this will cause "XREF" to overwrite them!

Also, under no circumstances should the high memory pointer be changed once the "XREF" program has been loaded. Failure to do this will cause problems with the "XREF" and "INSERT" commands.

After the "XREF" program has been loaded, go to BASIC and specify one file. After "XREF" is loaded, it displays the memory size which should be set from BASIC. The memory size need not be set (unless TRSDOS 2.1 or NEWDOOS 2.1 is used) if using NEWDOS/80, TRSDOS 2.3, LDOS or DOSPLUS since the XREF machine language program protects itself. If a MEMORY SIZE other that the one specified by "XREF" is set, it will cause problems when using the "XREF" and "INSERT" commands.

Next, type in:

```
RUN "ADVEDIT/BAS"
```

To modify or just peek (that's cheating!) at an existing data base, READ it into memory and then use the ADVEDIT commands to review it. To create a new adventure simply run ADVEDIT and start entering data via the MODIFY command.

The ADVEDIT commands and corresponding menu keys are listed below:

R	READ an adventure data base in.
W	WRITE an adventure data base out.
L	LIST the data base.
P	PRINT (hardcopy) the data base.
M	MODIFY a data base section.
I	INSERT blanks into the data base.
X	XREF: reference every occurrence of a data base section in the actions.
E	END the ADVEDIT program.

Most of these commands have options within them. A description of each command is given below.

READ command:

This command will read in an adventure data base. Simply supply the adventure number and the drive number. If the drive number is not entered, the first occurrence of the file is used.

The adventure number may be at most two characters from 0-9 and A-Z. Scott Adams' adventures 0-12 use the characters 0-9 and A-C (A for 10, B for 11, C for 12). This translates into a file name of "ADVENT/Dx" where "x" is the character 0-9 or A-C. If two characters are entered, such as "00" the file name would be "ADVENT/D00". The adventure number displayed by the adventure driver program (ADVENTUR/CMD or ADV/CMD) will be from 0-35. Only the first character of the file name is used in determining this number. A 0-9 are displayed as that number, an "A" is displayed as a 10, a "B" as an 11, etc. The TRAILER of the data base contains the adventure number. Scott Adams' adventure driver program requires this value to be a single digit from 0-9. Therefore, only the least significant digit of the adventure number is written out (0 as 0, 15 as 5, 23 as 3, etc.)

Examples of good file names are:

Characters	Resultingufile name
A	ADVENT/DA
0A	ADVENT/D0A
ZD	ADVENT/DZD

If an error occurs while reading a data base, an appropriate error message is displayed and control returns to the main menu.

If bad data somehow gets into the data base, a "*BAD

SECURITY*" message is displayed. The data base will have been read in, but the accuracy of the data can not be guaranteed.

WRITE command:

The WRITE command will store an adventure data base on disk. The adventure number and drive number are requested. If the adventure being written out was previously read in, hitting <ENTER> for the adventure number and drive number will write the adventure out with the same specifications.

The adventure number entered must be in the same format as in the READ command.

Before writing the data base, ADVEDIT verifies that the HEADER is holding the correct limiting values of the number of actions, messages, etc. For example, suppose you entered 50 messages but the HEADER said there were only 40. Writing the data base out without checking the limits would result in some lost data (messages 41-50). ADVEDIT makes a check and would write out all 50 messages.

LIST command:

This command is used to list on the CRT any part of the data base. After "L" is entered from the main menu a LIST sub-menu is displayed.

The options of the submenu are: Header, Action entries, Vocabulary, Rooms, Messages and Objects.

The section to be listed is selected by typing in the first letter of its name. For example, if "A" is pressed, the Action entries will be listed. If the "-" key is pressed, the LIST sub-menu is exited back to the main menu.

After a section of the data base is selected, the lower and upper limits to be displayed are input. If the ENTER key is depressed for this inquiry, all of that data base section will be listed. The listing will automatically pause after so many lines are displayed. To continue the listing, hit any key (except BREAK).

While the section is listing, hitting the SPACE BAR will cause the listing to be exited and the LIST sub-menu to be reentered. If the listing has paused, then pressing the SPACE BAR will not exit back to the LIST sub-menu.

The number of items in each data base section is kept in the

HEADER. This value is the upper limit used by ADVEDIT when the ENTER key is depressed on the limit inquiry (lower and upper bounds). The MODIFY command will allow input past this value without changing the value. As a result, all items of a data base section may not be reviewed on a LIST, PRINT or MODIFY. To fix this, just make sure the HEADER points to at least the highest value of the data base section in question.

To LIST Action entries 5 through 65, input the following (user inputs are underlined):

L   (Hit the "L" key from the main menu to enter the LIST sub-menu)

The computer will display the following:

```
Which section of the data base do you want to list:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M, O or - ? _
```

To select a data base section, hit the first letter of that section. In our example this would be the "A" key:

A   (To select the Action entries)

The computer will display:

```
  Lower Limit, Upper limit (ENTER is All) ? _
```

Now type in the limits (5 and 65):

5,65

The action entries should be listed starting at entry 5. After six have been listed, hit any key to continue listing. If the SPACE BAR is pressed while the section is listing, the LIST sub-menu will be reentered.

PRINT command:

The PRINT command will give a hardcopy listing of any one section or all of the data base. Hitting the "P" key while at the main menu will enter the PRINT sub-menu.

The options of the PRINT sub-menu are:

Everything, Header, Actions, Vocab, Messages, Rooms or Objects.

The PRINT sub-menu gives the option of printing any or all of the data base. Hitting the SPACE BAR while a section is being printed will return control to the PRINT sub-menu. The PRINT command, unlike the LIST command, gives no options for upper and lower bounds of printing. Hitting any of the PRINT sub-menu options will cause that data base section to be printed. The section is selected by pressing the first character of its name.

When the "P" key is pressed from the main menu the computer will display the following:

```
Do you want to print Everything in the data base
or just the Header, Actions, Vocab, Messages, Rooms or
Objects
  Type:  E, H, A, V, R, M, O or - ? _
```

If the "-" key is pressed, the main menu is reentered.

If any section of the data base has been entered past its limiting value (the value held in the HEADER) then all of the data will not be printed. To fix this, make sure the HEADER points to at least the highest value of the data base section being PRINTed.

MODIFY command:

The MODIFY command is used to edit an adventure. To enter the MODIFY sub-menu hit the "M" key while in the main menu.

The computer will display the following:

```
Which section do you want to modify:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M, O or - ? _
```

To modify a section of the data base, simply key in the first letter of its name. If the option selected is anything other than the HEADER, an inquiry is made for the lower and upper limits of the data base section to be modified. Hitting ENTER here will let the user modify all elements of that section. One note however, the user can MODIFY past the limit value held in the HEADER for each data base section. If ENTER is hit for the lower and upper bounds inquiry, any elements above the upper limit in the HEADER will be missed. The fix is to make sure the HEADER points to at least the last item in each data base section.

When modifying any section of the data base, hitting the ENTER key as a response will leave the item the same.

When an Action is modified, and all conditions and commands have been entered, an inquiry is made for "Y,N,-". The "Y" means the modified action is correct, "N" means it is not.

If "Y" is pressed, any changes to that action are stored and the next action entry is displayed. If the "N" key was pressed, the action entry will be modified again. If the "-" key is pressed, the action entry is assumed to be correct and the MODIFY sub-menu is reentered.

Also, when modifying actions, the conditions are entered with a comma between the word and the number. For example, "AVL,50" is a legal input.

Verbs and nouns must be separated by a comma (VERB,NOUN). Verbs and nouns input into actions must match exactly with ones found in the vocabulary or an error message will be printed. For example, if EXAMINE was the entry in the vocabulary list, EXAMINE would have to be the entry in the action (EXAM would not work).

Suppose an action entry was to have "LIGHT TORCH" as its verb-noun, the conditions were that object 10 must be being carried (object 10 is an unlit torch) and the commands would be to switch the location of the lit torch with the unlit one. The entry of the action would go as follows (underlined entries are input by the player):

```
M    (from the main menu to enter the MODIFY sub-menu)

Which section do you want to modify:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M, O or - ? A

(Select the Actions)

  Lower Limit, Upper limit (ENTER is All) ? 123,123
(Select Action 123)

Action 123:        Verb, Noun ? LIGHT,TORCH <ENTER>
PAR 0  Cond, Value ? HAS,10
PAR 0  Cond, Value ? PAR,10
PAR 0  Cond, Value ? PAR,9
PAR 0  Cond, Value ? <ENTER>
PAR 0  Cond, Value ? <ENTER>
 0  Cmd or Msg # ? EXX,X
 0  Cmd or Msg # ? <ENTER>
 0  Cmd or Msg # ? <ENTER>
 0  Cmd or Msg # ? <ENTER>
  Title ? <ENTER>
OK?  Type:  Y, N or - ? Y
```

See chapters 2 and 5 for more information on what the conditions and commands entered here actually do. The "PAR 0" and "0" preceding the inputs are the previous values of those entries. A "PAR,0" is entered to delete a condition. A "0" is entered to delete a command.

To enter a room called "CLOAK ROOM" with a north exit to room number 10 the following would be entered:

```
M    (from the main menu to enter the MODIFY sub-menu)

Which section do you want to modify:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M, O or - ? R

(Select the ROOMs)

  Lower Limit, Upper limit (ENTER is All) ? 5,5

(Select room number 5)

Room  5:    0 N    0 S    0 E    0 W    0 U    0 D
  Room description:
N,S,E,W,U,D rooms ? 10,0,0,0,0,0
Description ? Cloak room
```

The numbers preceding the letters (N, S, E, etc.) are the previous adjacent room numbers.

INSERT command:

The INSERT command will insert blank lines into certain data base sections. Hit the "I" key from the main menu to enter the INSERT sub-menu.

Insertions can be made into actions, verbs and nouns. To select a section, type in its first letter. Next, indicate the number of blank lines to be inserted. Lastly, respond with the item number the blank lines are to be inserted after. Hitting the "-" key returns control to the main menu.

The number of blank lines to be inserted must be a positive number. If it is not, an error message is displayed and control returns to the INSERT sub-menu.

A couple of applications of INSERT are given below:

At times, the user would like to add a synonym to an existing verb or noun in the vocabulary list. If there is not a blank line for the synonym after the primary verb or noun, the vocabulary has to be moved around. This is a real hassle since the action entries will probably have to be modified also. For example, if verb number 20 is moved to verb number 58, then all occurrences of verb 20 in the actions will have to be changed to verb 58. This is done by modifying the actions and retyping the verb-noun combination. An easy way to find all occurrences of a verb or noun is use the XREF command (discussed below) to find which actions they are used in.

Suppose the word EXAM was in the verb list and the synonym *HIT was put in its place. Every action entry with the verb EXAM would now have *HIT in its place. The easy way to add the synonym would be to insert a blank line and type the synonym in. The INSERT command could be used to put one blank line before EXAM (actually the blank line would be inserted after the verb just before EXAM) so *HIT could be entered, thus saving a lot of modifying. Inserting into nouns is done the same way.

There are some rules which must be followed however. No insertions may be made before verb 18 (DROP). Since an insertion before the DROP verb would move it from its predefined position, an insertion before it is not allowed.

No insertions are allowed before noun 6. Nouns 1-6 are the predefined room directions and they can not have any synonyms so no insertions may be done in them.

Also, at times a few blank lines may be needed between two action entries. For example, maybe an entry has to be lengthened to the point that a CONT command in the action entry must be added. If another action entry directly follows the entry to be CONTinued, at least one of them will have to be moved and retyped. The INSERT command will allow one or more blank lines to be inserted so the CONTinued action entry may be entered with no retyping.

Another use for the INSERT command on actions is for making more space for the automatic actions. The automatic action entries must precede all player input actions. If there is no more room for auto-actions, the INSERT command can make room by inserting some blank lines before the player input actions.

An example of INSERTing follows. Suppose you have the following partial noun list:

```
23:  SHELF
24:  *BOOK
25:  CASSETTE
26:  DISKETTE
27:  DOG
```

If you wanted to add the synonym "*TAPE" to the noun "CASSETTE" the INSERT command could be used. The procedure goes as follows (all user input is underlined):

Hit the "I" key from the main menu to enter the INSERT sub-menu. This message will be displayed:

```
What section of the data base do you want to insert into:
Actions, Verbs or Nouns
  Type:  A, V, N or - ? N
```

(The "N" was hit to select the nouns)

The computer will respond with:

```
How many blank lines ? 1
```

(We need to insert only one blank line)

The computer will inquire:

```
After what noun # ? 25
```

After a few seconds the insertion will be completed. After the task is finished the nouns will be listed as follows:

```
23:  SHELF
24:  *BOOK
25:  CASSETTE
26:
27:  DISKETTE
28:  DOG
```

Now the MODIFY command could be used to place the noun synonym "*TAPE" at noun 26.

As you can see, noun 26 (DISKETTE) was moved to noun 27 (all nouns following the selected noun are moved down). The INSERT command changes all affected action entries by this operation (instead of referring to noun 26, refer to noun 27).

One warning about INSERT. When an insertion is made, the highest item in the data base section selected will be moved up in the list. The limiting value in the HEADER is not updated however. A check should be made to determine if this is the case.

Most of the code of the INSERT command is contained in the machine language file "XREF/CMD". If this file is not loaded before ADVEDIT is run do not use the INSERT command.

XREF command:

The XREF command returns the number of every action entry a noun, verb, room, message, object, bit flag or counter appears in. To use this command type "X" from the main menu.

The XREF sub-menu will be printed on the screen. Hit the first letter of the section the XREF is to be run on. Enter the item number for the XREF (for example, which object number). ADVEDIT will not allow illegal values to be input.

Output may be routed to the screen or printer by typing an "S" or "P" when asked.

Finally, the limits of the actions the XREF is to be run on is entered. For example, you may want to see which actions from action entry 5 through action entry 134 reference object 34.

There are a few oddities about the XREF command. An XREF can not be done on message 0. It may also give strange results on nouns. XREF can find every occurrence of a noun number. However, no distinction is made from nouns and auto action probabilities. Just list the actions after the references are made so a distinction can be made.

When an XREF is run on the verbs, nouns or messages, a match is displayed by the message "ACTION: n" where "n" is the action number that particular item was found in.

When an XREF is run on an object, room, bit flag or counter one of two messages will be displayed.

The first one is "CONDITION - ACTION: n". This message is displayed if the item found was referenced in the conditions of the action. For example, "BIT 30", "HAS 10" and "IN 3" are references in the conditions.

The second one is "COMMAND - ACTION: n". This message is displayed if the item found was referenced in the commands of the action. For example, "GOTOY" where the associated parameter in the conditions was a "PAR 3" or "AGETX" where "PAR 10" was in the conditions.

When displaying matches, XREF does not pause the output. Therefore, if there are more than 15 or so matches, the first ones may be hard to read. In this case, select a smaller range of actions to be scanned and do the XREF in more than one part. Another solution is to send the output to the printer.

The following table tells what conditions are searched for when doing an XREF on the appropriate data base section:

OBJECTS	ROOMS	BIT FLAGS
HAS	IN	BIT
IN/W	-IN	-BIT
AVL		
-IN/W		

```
-HAVE
-AVL
-RMO
 RMO
 ORIG
-ORIG
```

The following table tells what commands are checked for when doing an XREF on the appropriate data base section:

OBJECTS	ROOMS	BIT FLAGS	COUNTERS
GETX	GOTOY	SETZ	EXM,CT
DROPX	X->Y	CLRZ	
X-RMO			
X->Y			
EXX,X			
AGETX			
BYX->X			

As you can see by the above table, the only command searched for when doing an XREF on a counter is the "EXM,CT" command. The reason for searching for only this command even though others may affect the counter (i.e. CT-1) is that there is no way of checking if the other commands (such as CT-1) are affecting that particular counter.

If must be noted that for every "EXM,CT" command for a particular counter, there will almost certainly be another "EXM,CT" command to switch it back in either the same action or in one closely following the first one. This situation must also be considered.

Suppose you wanted to find which actions referenced object number 25. The procedure would go as follows (user inputs are underlined):

From the main menu hit the "X" key to enter the XREF sub-menu:

```
What section of the data base do you want an XREF of:
Verbs, Nouns, Rooms, Messages, Objects, Bit flags or
Counters
  Type: V, N, R, M, O, B, C or - ? O
```

(Select the OBJECT XREF)

The computer will inquire:

```
Object # ? 25
```

(Select object 25)

```
Screen or Printer output:  Type S or P ? S

(Direct all output to the screen)

Actions scanned:
 Lower Limit, Upper limit (ENTER is All) ? <ENTER>

(Scan all of the actions)

CONDITION - ACTION:    6
COMMAND - ACTION:   45
CONDITION - ACTION:  123
```

The output in this example was contrived, but it indicates that object number 25 is referenced in the conditions of actions 6 and 123 and is referenced in the commands of action 45.

One warning about the XREF command. Most of the code for this command is in the machine language file "XREF/CMD". If this file is not loaded before the ADVEDIT program is run, the XREF command should not be used.

If the ENTER key is pressed for the range of actions scanned, the upper limit scanned will be the upper limit held in the HEADER. If the HEADER value is not high enough to encompass all actions, any actions above the HEADER value will not be checked.

END command:

Pressing the "E" key from the main menu causes the END command to be executed. The END command simply returns control to BASIC.

If the BREAK key is depressed at any time and the ADVEDIT program is halted, the instruction "GOTO 4" may be entered to resume operation with all data left intact.

Because of all the string usage in ADVEDIT, string compression may occasionally occur. This will result in pauses up to 30 seconds. In most cases, string compression will never pop up.

ADVEDIT has been checked out with NEWDOS/80 version 2.0, TRSDOS 2.2 and 2.3, NEWDOS/21, DOSPLUS, LDOS and NEWDOS/80 version 1.0 with success. The only problem occurred when using NEWDOS/80 version 1.0. If using this DOS, the zaps sent out by Apparat must be installed. Also, when entering message numbers in the action entries, the values must be two digit numbers. For example, message 5 is entered as "05" or " 5".

ADVEDIT LIMITATIONS

The following limitations are imposed on data entered via the ADVEDIT program:

1) Maximum number of action entries is 300.
2) Maximum number of vocabulary entries is 150.
3) Maximum number of rooms is 100.
4) Maximum number of messages is 100.
5) Maximum number of objects is 150.
6) Maximum characters in a text description of an object, room, message or action title is 255.

These "limitations" are far in excess in most cases of any current Scott Adams' Adventure.

The maximum values encountered by any current Scott Adams' adventure are:

270	Action entries
80	Vocabulary words
30	Rooms
99	Messages
100	Objects

Another limitation of ADVEDIT is that only integer numbers may be entered. If a number out of the range of +32767 to -32768 is entered, an error will occur and an error message will be displayed. In fact, if any BASIC error occurs, the BASIC error number will be displayed and control will return to the main menu.

Suggested entry for ADVEDIT

The first step in using ADVEDIT is writing the adventure on paper. It is sort of like writing a story. Just the basic idea of the adventure is needed. For example, in Scott Adams' Adventure 3 (Mission Impossible), the basic idea is to disarm a saboteur's time bomb planted in the core of a nuclear reactor.

After the basic idea is down on paper, some of the finer details should be considered. For example, in Adventure 3 there are three doors that require some sort of identification to get through them.

After the finer details are done, start writing down the rooms, vocabulary and objects on paper. After most of these are entered, start writing the actions and messages. The reason for writing the actions and messages last is so you know what objects and rooms you have to work with and can add more when needed.

After the adventure is roughly down on paper, start entering it into the ADVEDIT program. The HEADER should be modified first. The values input here do not have to be exact, just approximate.

A big decision has to be made at this time. That is the word length of the adventure. Scott Adams uses 3 or 4 letters for the word length on his adventures. The word length is significant because the length of the object names (identifier between slashes for objects to be picked and dropped) must match this value or be shorter.

After the HEADER is entered, the order of section entry does not matter (except that the actions must be entered after the vocabulary).

After the adventure is typed in, double check that no text descriptions contain quote marks. If they do, change them to SHIFTed @ signs (ADVEDIT does this in all known cases).

Next, double check that the header values are high enough to contain all of the data (i.e. the number of actions in the HEADER is at least as large as the actual number of actions).

Lastly, save the adventure out with the WRITE command and END the program.

Now go to DOS and run Scott Adams' ADVENTUR program or Bruce Hansen's ADV program. The adventure number to be entered must be the same as the one used when writing the adventure out. Scott Adams' ADVENTURE program only allows one character to be input (0-9, A-Z). If the adventure data base was written out with a two character file name and Scott's driver is to be used, it must be RENAMEd from DOS to one with a single character name (for example, ADVENT/DA not ADVENT/DAA).

Now comes the time consuming part, debugging the adventure. The way to do this is check every possible thing you can think of to see if the actions are performing the way you wanted them to. If they don't, take notes on which ones aren't working and continue playing your ADVENTURE. After finishing with this procedure, run the ADVEDIT program again and fix any errors. This process takes longer than writing the adventure in most cases. But after much hard work you should have an adventure you could be proud of.

Chapter 5

Sample Adventure

This chapter explains, in depth, a short adventure entitled "MINI-VENTURE." This adventure uses most of the conditions and commands available in ADVENTURE. Every significant part of the data base is explained. But first here's a listing of the data base:

```
Adventure  35  Version  1.01  7837 bytes under 16K (tape) or
11214
under 32K (disk)
BYTES  #OBJ  #ACT  #VOC  #RM CARRY START #TR WLEN TIME  #MSG
TR-RM
1008    14    41    22    8    5    1    1    4   999   16
  7


ACTIONS
# --   V     N     COND 1   COND 2  COND 3  COND 4  COND 5
          CMD 1    CMD 2   CMD 3   CMD 4   ACTION TITLE ---
 0: AUTO   100    -BIT 1   PAR 1      0       0       0
          MSG1     SETZ      -       -     INTRO
 1: AUTO   100    -IN 2    PAR 1   PAR 4   PAR 1      0
          EXM,CT   CT<-N   EXM,CT    -     SET MUG CNTR
 2: AUTO   100    IN 2     PAR 1      0       0       0
          EXM,CT   CT-1    CONT      -     MUGGED?
 3: AUTO     0    CT= 0       0       0       0       0
          MSG2     DEAD    FINI      -
 4: AUTO     0    PAR 1       0       0       0       0
          EXM,CT    -        -       -
 5: AUTO   100    -IN 2    BIT 15     0       0       0
          DAY      DSPRM     -       -     IN LIGHT?
 6: AUTO   100    IN 2    -BIT 15     0       0       0
          NIGHT    DSPRM     -       -     IN DARK?
 7: LIGH   MATC   HAS 13   PAR 9      0       0       0
          AGETX    DSPRM   MSG3    CONT    LIGHT MATCH
 8: AUTO     0    PAR 9       0       0       0       0
          DELAY    DELAY   X->RMO  DSPRM
 9: AUTO     0       0        0       0       0       0
          MSG4      -        -       -
10: GET    KEY    IN 1     RMO 12  PAR 12     0       0
          MSG5     AGETX     -       -
11: GET    KEY    IN/W 12  PAR 12     0       0       0
          GETX     MSG5      -       -
12: DROP   KEY    HAS 12   PAR 12     0       0       0
          DROPX    MSG5      -       -
13: EXAM   KEY    AVL 12      0       0       0       0
          MSG6      -        -       -
14: GO     DOOR   IN 2     PAR 3      0       0       0
          GOTOY    MSG5      -       -
15: UNLO   DOOR   HAS 12   IN/W 3     0       0       0
```

```
           MSG7         -          -          -
16: EXAM   DOOR   IN/W 3        0          0          0          0
           MSG8   MSG9          -          -
17: EXAM   WHEE   IN 1      RMO 12         0          0          0
           MSG12        -          -          -
18: GO     ELEV   IN/W 4    PAR 4          0          0          0
           GOTOY  MSG5          -          -
19: GO     ELEV   IN/W 6    PAR 4          0          0          0
           GOTOY  MSG5          -          -
20: EXAM   PANE   IN/W 5        0          0          0          0
           MSG13        -          -          -
21: PUSH   1      IN 4      PAR 2          0          0          0
           CLRZ   MSG5          -          -
22: PUSH   2      IN 4      PAR 2          0          0          0
           SETZ   MSG5          -          -
23: GO     ROOM   IN 4      -BIT 2    PAR 3           0          0
           GOTOY  MSG5          -          -
24: GO     ROOM   IN 4      BIT 2     PAR 5           0          0
           GOTOY  MSG5          -          -
25: UNLO   DOOR   IN/W 7    HAS 12         0          0          0
           MSG7         -          -          -
26: EXAM   DOOR   IN/W 7        0          0          0          0
           MSG8   MSG10         -          -
27: EXAM   DOOR   IN/W 8        0          0          0          0
           MSG8   MSG11         -          -
28: EXAM   DOOR   IN/W 10       0          0          0          0
           MSG8   MSG11         -          -
29: UNLO   DOOR   HAS 12    IN/W 8    PAR 8     PAR 10           0
           EXX,X  MSG5          -          -
30: LOCK   DOOR   HAS 12    IN/W 10   PAR 8     PAR 10           0
           EXX,X  MSG5          -          -
31: GO     DOOR   IN/W 10   PAR 7          0          0          0
           GOTOY  MSG5          -          -
32: SAVE   GAME         0          0          0          0          0
           SAVE         -          -          -
33: QUIT   ANY          0          0          0          0          0
           FINI         -          -          -
34: SCOR   ANY          0          0          0          0          0
           SCORE        -          -          -
35: INVE   ANY          0          0          0          0          0
           INV          -          -          -
36: EXAM   ANY          0          0          0          0          0
           MSG14        -          -          -
37: HELP   ANY          0          0          0          0          0
           MSG15        -          -          -
38: GO     CAR    IN 2      PAR 1          0          0          0
           GOTOY  MSG5          -          -
39: PUSH   BUTT         0          0          0          0          0
           MSG16        -          -          -
40: TURN   ANY          0          0          0          0          0
           MSG5         -          -          -
41: AUTO         0      0          0          0          0          0
                 -      -          -          -
```

```
VOCABULARY
# --- VERBS    NOUNS ---
 0:   AUTO     ANY
 1:   GO       NORTH
 2:   *ENTE    SOUTH
 3:   *WALK    EAST
 4:   EXAM     WEST
 5:   *LOOK    UP
 6:   LIGH     DOWN
 7:   HELP     DOOR
 8:   UNLO     KEY
 9:   LOCK     *KEYS
10:   GET      ROOM
11:   *TAKE    BUTT
12:   *PULL    CAR
13:   PUSH     1
14:   INVE     2
15:   QUIT     ELEV
16:   SAVE     PANE
17:   TURN     GAME
18:   DROP     WALL
19:   SCOR     WHEE
20:            MATC
21:
22:

ROOMS
# ---    N    S    E    W    U    D    ROOM DESCRIPTION ---
 0:      0    0    0    0    0    0   Storeroom.  Can't get here
 1:      0    0    2    0    0    0   car with an open door
 2:      0    0    0    0    0    0   *I'm on the curb
 3:      0    2    0    0    0    0   hallway
 4:      0    0    0    0    0    0   *I'm in an elevator next to  a
room
 5:      0    0    6    0    0    0   hallway
 6:      0    0    0    5    0    0   hallway
 7:      0    0    6    0    0    0   *I'm in my apartment
 8:      0    0    0    0    0    0   hospital

MESSAGES
# --- MESSAGE TEXT ---
 0:
 1: Welcome to "MINI-VENTURE" by Bruce Hansen
 2: I was MUGGED!!
 3: The match flares up
 4: and goes out.
 5: OK
 6: The number "201" is stamped on one of them
 7: The key won't fit
 8: There's a plate with
 9: 101 on it
10: 200 on it
11: 201 on it
12: There's a set of keys in the ignition
```

```
13: There are two buttons marked "1" and "2"
14: I see nothing special
15: HOW?
16: Say again with which button

OBJECTS
# --- START   OBJECT DESCRIPTION ---
 0:    -1    *MY WALLET*/WALL/
 1:     1    Steering wheel
 2:     2    Apartment complex main door
 3:     3    Locked apartment door
 4:     3    Elevator
 5:     4    Elevator panel
 6:     5    Elevator
 7:     5    Locked apartment door
 8:     6    Locked apartment door
 9:     0    Lighted artificial light source
10:     0    Open apartment door
11:     7    Sign saying "LEAVE *TREASURES* HERE"
12:     0    Keys/KEY/
13:    -1    Matches/MATC/
14:     2    Car
```

The data base will be explained one section at a time. First the HEADER:

Adventure 35 is the adventure number (Z in this case). The version # is 1.01 and the adventure leaves 7837 bytes free in a 16K machine (tape version) and 11214 bytes free in a 32K machine (disk version). These memory values are the free bytes when using Scott Adams' ADVENTURE program. Bruce Hansen's ADV program uses approximately 600 fewer bytes. BYTES=1008 is the number of bytes this adventure uses, #OBJ=14 means there are 14 objects, #ACT=41 means there are 41 actions, #VOC=22 means there are 22 verbs and 22 nouns, #RM=8 means there are 8 rooms, CARRY=5 means the adventurer can carry a maximum of 5 objects, START=1 means the player starts in room 1, #TR=1 means there is only 1 treasure, WLEN=4 means the number of significant letters in the nouns and verbs is 4, TIME=999 means the light limit is 999 moves, #MSG=16 means there are 16 messages and TR-RM=7 means the treasure room is room 7.

ACTIONS:

The automatic actions must be placed before player input actions. The AUTO verb signifies an auto action. The noun is the probability of this action being considered. ADVENTURE does not scan all player input actions, just until it finds a true one (if one exists). All auto actions are considered even if a previous one is true.

This description does not tell what the messages, objects and rooms are (i.e. their word description) in most cases so refer to the above data base listing for that information.

```
0: AUTO 100   -BIT 1  PAR 1    0      0      0
      MSG 1     SETZ        -        -    INTRO
```

The 0: means this is ACTION 0. The AUTO 100 means this auto action is considered 100 percent of the time. The -BIT 1 means if bit flag 1 is cleared, this condition is true. When ADVENTURE is started, all bit flags are cleared so on the first pass of the auto actions this condition will be true. This is useful for printing introductions. The PAR 1 is a parameter to be passed to the commands if all conditions are met. If all conditions are met then the commands are executed. In this case message 1 would be printed and bit flag 1 would be set (SETZ). Bit flag 1 is set since the first parameter in the conditions was a 1. The word INTRO is an optional action title. Message 1 is the introduction message.

```
1: AUTO 100   -IN 2   PAR 1   PAR 4   PAR 1   0
     EXM,CT     CT<-N     EXM,CT    -    SET MUG CNTR
```

This is an automatic action used to set a counter. In this adventure if the player is outside his car and not in the apartment building for 4 consecutive moves he is mugged and killed (he loses). This action is executed 100 percent of the time (AUTO 100). The condition -IN 2 will be true if the player is in any room other than room 2. PAR 1, PAR 4 and PAR 1 are parameters used by the commands if all conditions are true. If all conditions are true, the CT (counter) value is exchanged (EXM,CT) with alternate counter 1 since the first parameter in the conditions was a 1. The command CT<-N will set the counter to 4 (4 is the next parameter). And finally CT is exchanged back with alternate counter 1 (EXM,CT). The reason for this switching is that ADVENTURE can only operate directly on the CT variable. ADVENTURE can operate on a maximum of 9 additional alternate counters however. SET MUG CNTR is the optional action title.

```
2: AUTO 100   IN 2    PAR 1    0       0      0
     EXM,CT     CT-1      CONT       -    MUGGED?
```

This condition is considered 100 percent of the time. The condition IN 2 passes if the player is in room 2. PAR 1 is a parameter used in the commands. If all conditions are true the commands are executed. In this case CT is exchanged with alternate counter 1 (EXM,CT) since the first parameter found was a 1. Then CT is decremented (CT-1). CONT means to continue considering all following AUTO 0's until an AUTO 1-100 or a player input action is found. In this case the next two actions are AUTO 0's. There are as follows:

```
3: AUTO  0   CT= 0   0      0      0       0
     MSG 2     DEAD     FINI     -
4: AUTO  0   PAR 1   0      0      0       0
     EXM,CT    -        -        -
```

In action 3, CT is tested to see if it is equal to 0 (CT= 0). If it does, then message 2 (MSG 2) is printed, the player is killed (DEAD) and the game is finished (FINI). If the counter was not equal to zero then action 4 is considered. It would normally be considered regardless of the pass/no pass status of action 3. But in this case since the DEAD command was issued the player was moved to the last room and killed. The FINI command then halted the game. This halts the auto actions. Since action 4 has no conditions it is always true and PAR 1 is passed to the commands. In this case, CT is exchanged with counter 1 again thus putting them back in their starting positions. There is a good reason for making these two actions separate. Two tasks need to be done, check if the counter is equal to zero and switch CT back with alternate counter 1. If these two were put in the same action and the counter (CT) did not equal zero, then the commands would not be performed. In this case the EXM,CT command would not be done so the counters would not be returned to the right place.

```
5: AUTO 100  -IN 2   BIT 15   0      0      0
     DAY     DSPRM      -       -     IN LIGHT?
```

This auto action is considered 100 percent of the time. The conditions are -IN 2 and BIT 15. -IN 2 is true if the player is not in room 2. BIT 15 passes if it is dark (bit flag 15 is defined by ADVENTURE for light/dark status). If the conditions are true, then the commands DAY and DSPRM are executed. The DAY command makes it day and DSPRM displays the current room. The reason for this action is to make it day after the player is off the curb (where it is dark). The BIT 15 condition is included so it is made DAY only when it was just NIGHT. The reason for this is that the DSPRM command makes the screen "flicker" when it is executed.

```
6: AUTO 100  IN 2    -BIT 15  0      0      0
      NIGHT     DSPRM      -       -    IN DARK?
```

This action is considered 100 percent of the time. If the player is in room 2 (IN 2) and bit flag 15 is cleared (-BIT 15) then the commands are performed. NIGHT makes it dark out (if the player is not holding the artificial light source) and DSPRM displays the room. This action makes it dark when the player is on the curb (room 2) and only does a NIGHT and DSPRM if it was previously DAY. The reason for not doing a DSPRM every time is that it causes the screen to be redrawn, which makes it look like the screen just glitched.

```
7: LIGH MATC HAS 13  PAR 9    0      0      0
```

```
      AGETX    DSPRM     MSG3   CONT LIGHT MATCH
```
This is the first player input action. If the player types in LIGHT MATCH then this action is considered. If even one of the conditions of this action are not met, then ADVENTURE continues searching the actions for another LIGHT MATCH. If it finds another, it considers that one also, and so on until it finds a true one. If no other LIGHT MATCH is found then the message "I can't do that . . . yet!" is printed and the player is asked to respond again. The condition in this action is HAS 13. So if the player HAS 13 (has the matches - object 13) the commands are performed. AGETX will make object 9 (from PAR 9 in the conditions) be carried by the player regardless of the carry limit. DSPRM will make it day if it is currently light out or object 9 is available (the artificial light source). Since an AGET 9 was just executed, the DSPRM will make it light if it was dark out or leave it light if it was light. Message 3 is then printed (MSG 3) and the CONTinue flag is set. All following AUTO 0 actions will be considered. These actions are as follows:

```
8: AUTO  0   PAR 9   0      0       0       0
      DELAY    DELAY    X->RMO   DSPRM
9: AUTO  0    0      0      0       0       0
      MSG 4      -        -        -
```
Action 8 has no conditions so its commands are executed. A DELAY command makes the program stall for about 1 second. After two such stalls, the PAR 9 object is put back in RMO (X->RMO - PAR 9 is the first parameter from the conditions). After the artificial light source is removed from the room another DSPRM is executed. This will make it dark again if it was dark before the match was lit or light if it was light before the match was lit. Action 9 is considered next. Since it has no conditions its commands are performed. In this case message 4 is printed. Since ADVENTURE found a matching player input action it does not consider any following player input actions and now checks the automatic actions (ones at the beginning of the actions).

```
10: GET  KEY  IN 1   RMO 12   PAR 12   0     0
      MSG 5    AGETX      -        -
```
If the player types in GET KEY this action is considered. It passes if the player is in room 1 (IN 1) and object 12 is in room 0 (RMO 12). If these conditions are true, then message 5 is printed (MSG 5) and the player is forced to pick up object 12 (AGETX - PAR 12 passed from conditions). The logic of this action is simple. The player must be in the car (IN 1) and not have already gotten the keys some time before (RMO 12) for the player to be able to get the keys.

```
11: GET  KEY  IN/W 12 PAR 12   0     0      0
      GETX     MSG 5      -        -
```
If action 10 failed for any reason then this action is

considered since they both have the same verb-noun combination. In this action the player is allowed to pick up object 12 if he is in with, but not carrying, object 12 (IN/W 12). If true, a GETX command is performed and message 5 is printed (MSG 5). A GETX checks to see if the carry limit is exceeded. If not the player picks up object 12 (GETX - PAR 12 from the conditions). This action would not be needed if action 10 was not included. Objects which are named (/name/ at the end of the object description) can normally be picked up and dropped without this type of action. But by having a GET action for object 12, the automatic GET feature for that particular object is disabled. In this case, GET has to be included in the actions.

```
12: DROP KEY  HAS 12  PAR 12  0      0      0
      DROPX     MSG 5     -      -
```
If the player is carrying object 12 (HAS 12) then object 12 is dropped (DROPX - PAR 12 from the conditions).

```
13: EXAM KEY  AVL 12  0       0      0      0
      MSG6      -         -      -
```
If object 12 is either being carried or is in the same room as the player (AVL 12), then message 6 is printed (MSG6). This type of action is very common for such things as examining, reading, etc.

```
14: GO   DOOR IN 2    PAR 3   0      0      0
      GOTOY     MSG5      -      -
```
If the player is in room 2 (IN 2) then the player is sent to room 3 (GOTOY - PAR 3 from the conditions) and message 5 is printed (MSG5). This is a common action for GOs without a direction. For example, GO CAVE, GO TUNNEL, etc.

```
15: UNLO DOOR HAS 12  IN/W 3  0      0      0
      MSG7      -         -      -
```
If the player types UNLOCK DOOR this action is considered. There are three doors in this adventure and the player has a key which will open only one of them. If the player has object 12 (HAS 12), and is in the same room as object 3 (IN/W 3) then message 7 is printed.

```
16: EXAM DOOR IN/W 3  0       0      0      0
      MSG8      MSG9      -      -
```
This is a common type of EXAMINE action. A message is printed when the object is examined, provided the player is by the object. The IN/W condition is usually used for an object which can not be carried. The AVL condition is used for an object which can be carried. In this case, if the player is in with object 3 (IN/W 3), then messages 8 and 9 are printed (MSG8 and MSG9). Note that message 8 contains only the first half of the EXAM message. The second half is message 9 for this door. Message 10 and message 11 are used for the second halves of the two doors which can not be

opened by the player (all three use the same first half of the EXAM message).

```
17: EXAM WHEE IN 1   RMO 12   0      0      0
      MSG12     -        -       -
```

This EXAMINE action has two conditions. If the player is in room 1 (IN 1) and object 12 is in room 0 (RMO 12) then message 12 is printed (MSG12). The logic for this action is as follows: if the player examines the wheel and the keys (object 12) are still there (RMO 12) then an appropriate message is printed. However, if the keys had been previously picked up then this EXAMINE would fail and ADVENTURE would search for another matching EXAM WHEE (or EXAM ANY).

```
18: GO  ELEV  IN/W 4 PAR 4    0      0      0
      GOTOY     MSG 5      -       -
19: GO  ELEV  IN/W 6 PAR 4    0      0      0
      GOTOY     MSG 5      -       -
```

These two actions are described together since they are very similar. There are 2 objects called "ELEVATOR" in this adventure. One is on the top floor, the other on the bottom floor. These actions check if the player is in with an elevator (IN/W 4 or IN/W 6) and if so sends the player to room 4 (GOTOY - PAR 4 from the conditions).

```
20: EXAM PANE IN/W 5  0      0      0      0
      MSG13     -        -       -
```

If the player is in with object 5 (IN/W 5) then message 13 is printed (MSG13). If the player is not in with object 5 ADVENTURE searches for another EXAM PANE.

```
21: PUSH 1    IN 4     PAR 2    0      0      0
      CLRZ      MSG5       -       -
```

This action uses a bit flag. If the player is in the elevator (IN 4) then bit flag 2 is cleared (CLRZ - PAR 2 from the conditions). When this adventure was written it was decided to use bit flag 2 cleared for floor "1" and bit flag 2 set for floor "2." Since the elevator would initially be at floor "1" this was logical because all bit flags are cleared at the start of ADVENTURE. The status of the bit flag is tested so ADVENTURE knows which floor of the apartment complex to send the player when he leaves the elevator. There are two different rooms adjacent to the elevator so this condition must be checked.

```
22: PUSH 2    IN 4     PAR 2    0      0      0
      SETZ      MSG5       -       -
```

This action is very similar to the above action except the bit flag is set, not cleared. This tells ADVENTURE to put the player on the 2nd floor when he leaves the elevator instead of the 1st floor.

```
23: GO  ROOM IN 4    -BIT 2  PAR 3   0      0
```

```
      GOTOY    MSG5      -      -
```
If the player is in room 4 (IN 4) and bit flag 2 is cleared (-BIT 2), then the player is sent to room 3 (GOTOY - PAR 3 from the conditions) and message 5 is printed (MSG 5). In words, if the player is in the elevator, and he last pushed "1", then he is sent to the 1st floor. If bit flag 2 is set, however, then this action will fail.

```
24: GO  ROOM IN 4    BIT 2    PAR 5  0      0
      GOTOY    MSG5      -      -
```
If the player is in room 4 (IN 4) and bit flag 2 is set (BIT 2), then the player is put in room 5 (GOTOY - PAR 5 from the conditions) and message 5 is printed (MSG 5). In words, if the player is in the elevator, and he last pushed "2", then he is sent to the 2nd floor.

```
25: UNLO DOOR IN/W 7 HAS 12   0      0      0
      MSG7      -        -      -
```
This action is the same as action 15 except this action is for a different door (object 7, not object 3).

```
26: EXAM DOOR IN/W 7  0      0      0      0
      MSG8     MSG10     -      -
27: EXAM DOOR IN/W 8  0      0      0      0
      MSG8     MSG11     -      -
28: EXAM DOOR IN/W 10 0      0      0      0
      MSG8     MSG11     -      -
```
These three actions are included together since they are very similar (the only difference is that they refer to different objects). Actions 27 and 28 really refer to the same thing. In action 27, the examine refers to a closed door in room 6. In action 28, the examine refers to the same door except it has now been opened so object 10 is in the room (open door) instead of object 8 (locked door). These actions are very similar to action 16.

```
29: UNLO DOOR HAS 12   IN/W 8   PAR 8   PAR 10   0
      EXX,X    MSG5      -      -
```
This is the only door the player can unlock. The door may be unlocked if the following conditions are true: the player is holding object 12 (HAS 12 - the keys) and he is in with object 8 (IN/W 8 - locked door). If these conditions are met then the room locations of object 8 and object 10 are exchanged (EXX,X - PAR 8 and PAR 10 from the conditions) and MSG5 is printed.

```
30: LOCK DOOR HAS 12 IN/W 10   PAR 8   PAR 10   0
      EXX,X    MSG5      -      -
```
This action is very similar to the previous one except this one locks the door. To lock the door these conditions must be met: the player is carrying object 12 (HAS 12 - the keys) and he is in with object 10 (IN/W 10 - open door). If the conditions are met, then object 8 and object 10 are exchanged (EXX,X - PAR 8 and PAR 10 from the conditions) and

MSG5 is printed.

```
31: GO   DOOR   IN/W 10 PAR 7    0      0      0
     GOTOY      MSG5       -      -
```
If the player wants to go through the door, this condition must be met: the player is in with object 10 (IN/W 10 - an open door). If that condition was met, the player is put in room 7 (GOTOY - PAR 7 from the conditions).

```
32: SAVE GAME  0      0      0      0      0
     SAVE      -             -      -
```
This action lets the player save the game. No conditions need be met so the SAVE command is always executed.

```
33: QUIT ANY   0      0      0      0      0
     FINI      -             -      -
```
This action lets the player stop the game. The ANY noun means that the action is considered if the player's input noun was any of the nouns in the vocabulary list. This action requires no conditions to be met and performs a FINI command.

```
34: SCOR ANY   0      0      0      0      0
     SCORE     -             -      -
```
This action will print the player's score. This message gives the number of treasures stored in the treasure room and the percent stored. No conditions are needed so the SCOR command is always performed.

```
35: INVE ANY   0      0      0      0      0
     INV       -             -      -
```
This command tells the player the name of every object he is currently carrying. The INV command is directly executed since no conditions are present.

```
36: EXAM ANY   0      0      0      0      0
     MSG14     -             -      -
```
This action will print message 14 (MSG14) if the object being examined was not referred to in a previous true examine action. This action is usually present in every adventure. The message printed is usually something like "I see nothing special."

```
37: HELP ANY   0      0      0      0      0
     MSG15     -             -      -
```
This is another common action. If the player types HELP message 15 (MSG15) is printed provided no previous HELP action was found to be true.

```
38: GO   CAR   IN 2   PAR 1    0      0      0
     GOTOY     MSG5          -      -
```
If the player is in room 2 (IN 2) then he is put in room 1 (GOTOY - PAR 1 from the conditions) and message 5 is printed (MSG5).

```
39: PUSH BUTT  0     0        0      0      0
     MSG16     -         -      -
```

If the player PUSHes BUTTon then message 16 (MSG16) is printed. Since the player is supposed to PUSH 1 or PUSH 2 this action is used to tell him so.

```
40: TURN ANY   0     0        0      0      0
     MSG5      -         -      -
```

If the player tries to TURN any legal object message 5 (MSG5) is printed.

```
41: AUTO 0     0     0        0      0      0
         -     -         -      -
```

This action is not used.

VOCABULARY

The user must refer to the VOCABULARY words in the data base list for this explanation. Notice that the predefined verbs and nouns are in their proper places (AUTO, GO, GET, DROP, ANY, NORTH, SOUTH, EAST, WEST, UP and DOWN). The only thing special about the vocabulary words is the synonyms. Part of the vocabulary appears as follows:

```
0:    AUTO
1:    GO
2:    *ENTE
3:    EXAM
4:    *LOOK
```

Synonyms are required to directly follow their primary noun or verb in the list and must be preceded by an asterisk. In this case *ENTE is a synonym of GO and *LOOK is a synonym of EXAM. There may be more than one synonym for a certain noun or verb (see verb 10 - GET). Any action entries using the nouns or verbs must refer to a primary noun or verb, not a synonym.

ROOMS

An example of two rooms follows:

```
#-- N  S  E  W  U  D  ROOM DESCRIPTION
6:  0  0  0  5  0  0  hallway
7:  0  0  6  0  0  0  *I'm in my apartment
```

The 6: and 7: are the room number (6 and 7). The next six numbers are the rooms which the player will move to if he goes in the corresponding direction. For example, if the player is in room 6 and he types GO WEST, ADVENTURE will send him to room 5. If in room 7, typing GO EAST will put the player in room 6. The zeros mean the player can't go in that direction.

The ROOM DESCRIPTIONs "hallway" and "*I'm in my apartment" are examples of the default room message. For example, if the player is in room 6, the room is described as "I'm in a hallway." If in room 7 the room is described as "I'm in my apartment." Putting an asterisk before the room description disables the automatic prefix message "I'm in a."

MESSAGES

The messages are just text strings. Look at message 1 however. In this listing the quote marks (") appear as quote marks. However, in BASIC these will appear as at signs (@) unless an upper/lower case modification has been installed. If an upper/lower case conversion has been installed, the character may look like a single quote or a British pound sign.

Notice that message 0 is not used. It should be null since it is printed by action entries when no other command is given.

OBJECTS

Each object has a starting room and a description. Objects which have a name, for example "Keys/KEY/" may be automatically picked up and dropped (The name is KEY in this case). If the starting room is -1 (like it is for *MY WALLET*) then the object is carried by the player when the adventure is started. Any other number is the room number in which that object can be found. A room number of zero means the object has not been found yet and is in the storeroom.

Notice that object 9 is titled "Lighted artificial light source." Object 9 is predefined by ADVENTURE as the artificial light source in its lighted condition. In this adventure, object 9 is not used (except for the LIGHT MATC action) so no real name is given to it.

These are the steps needed to win at this adventure:

1) GET the keys from the car (EXAM WHEE will tell the player if they are there).
2) Get out of the car by moving EAST.
3) To see, light a match.

4) Enter the apartment building by typing GO DOOR.
5) Type GO ELEV and PUSH 2 to go to the second floor.
6) GO ROOM to leave the elevator.
7) GO EAST to the second locked apartment door on the second floor.
8) UNLO DOOR to open the locked apartment door.
9) GO DOOR and drop the wallet (the wallet is initially carried by the player).
10) Type SCOR.

For a more "challenging" adventure try adventures "X" and "Y" on THE ADVENTURE SYSTEM master diskette. These adventures are considerably longer than this one and will pose much more of a challenge.

Chapter 6

Solving an ADVENTURE

This chapter will briefly describe a method for solving adventures using the ADVEDIT program.

There are two basic types of adventures: mission and treasure.

In mission adventures, the object of the game is to accomplish a task. In adventure 3, the task is to disarm a saboteur's time bomb. In adventure 4, the mission is to save Count Cristo.

In treasure adventures, the object of the game is to collect treasures and store them in the treasure room.

SOLVING "MISSION" TYPE ADVENTURES

Mission type adventures end with a winning message. The first step to solving these types of adventures is to list the messages and find the message number. This should be an obvious message. The message number should be noted.

Next, do an XREF for that message number in the actions. This procedure will tell you which actions display the winning message. The number(s) of these action(s) should be noted.

Now, list the action(s) containing the winning message. Note what the conditions are. These conditions must be true before the winning message will be displayed.

The XREF command can be used to find where the objects needed in the winning action are referenced. This will tell you how to get them if they are not simply laying in a room (where they could be picked up). If a bit flag needs to be set before the winning message is displayed, an XREF can be done on that particular bit flag to find out what must be done to set it.

The procedure continues in this fashion. It may take a while to get it down pat, but it is basically a simple procedure.

SOLVING "TREASURE" TYPE ADVENTURES

These types of adventures are very similar to mission types adventures when solving them.

The first step in solving them is to list the OBJECTS. Note which objects are treasures.

Treasures that have a non-zero room number are simply laying in a room. The only potential problem here is that some actions must need to be taken to get into the room. For example, a locked door may block the entrance of the room. By looking at the room descriptions, it can be determined if this room can be moved into from another room (for example, GO EAST from another room moves you into the one in question). If not, do an XREF to find what conditions must be true to enter the room.

If the treasure has a room number of zero, then some action must take place to drop it in a room. By doing an XREF on the treasure, it can be determined what conditions must be met for the treasure to enter a room so it may be picked up.

The procedure continues for all of the treasures.

It may take some time to solve an adventure by this method, but it is possible. In fact, the author solved Scott Adams' adventure 9 using this method.

However, the best way to solve an adventure is to play it through. If you get stuck, look at the data base as little as possible unless you're fed up with the adventure. Remember, adventures are meant to be brain-teasers.

Appendix A

ADVENTURE Command summary

CONDITIONS:

PAR	Passes a number to the commands.
HAS	True if holding the object.
IN/W	True if in same room as object (not holding it).
AVL	True if in same room or holding object.
IN	True if in room.
-IN/W	True if holding object or if object is in another room.
-HAVE	True if not holding object.
-IN	True if not in room.
BIT	True if bit flag set.
-BIT	True if bit flag cleared.
ANY	True if holding any objects.
-ANY	True if not holding any objects.
-AVL	True if object in another room.
-RMO	True if object not in room zero.
RMO	True if object in room zero.
CT<=	True if counter less than or equal to number.
CT>	True if counter greater than number.
ORIG	True if object in original starting room.
-ORIG	True if object not in original starting room.
CT=	True if counter equal to number.

Commands:

GETX	Pick up object X.
DROPX	Drop object X.
GOTOY	Move player to room Y.
X->RMO	Send object X to room zero.
NIGHT	Make it night (set bit flag 15).
DAY	Make it day (clear bit flag 15).
SETZ	Set bit flag Z.
CLRZ	Clear bit flag Z.
DEAD	Tell player he's dead, make DAY, move to last room, end game.
X->Y	Send object X to room Y.
FINI	Stop game and ask for another game.
DSPRM	Display current room and account for DAY, NIGHT.
SCORE	Compute the score.
INV	Tell the player what he is carrying.
SETO	Set bit flag 0.
CLRO	Clear bit flag 0.
FILL	Fill artificial light source (clear bit flag 16).
SAVE	Save the game.
EXX,X	Exchange room location of object X with object X.
CONT	Continue to next action/s.
AGETX	Always get object X regardless of carry limit status.

BYX->X	Move second object X to same place as first object X.
CT-1	Decrement counter.
DSPCT	Display the counter.
CT<-N	Set counter equal to N.
EXRM0	Exchange current room with room held in alternate room register 0.
EXM,CT	Exchange counter and alternate counter M.
CT+N	Add N to counter.
CT-N	Subtract N from counter.
SAYW	Say the player's input noun.
SAYWCR	Say the noun of the player's input noun and a carriage return.
SAYCR	Start a new line.
EXC,CR	Exchange current room with room in alternate room register C.
DELAY	Pause for about 1 second.

Appendix B

Submitting your adventures for marketing considerations

All you need to do is send a diskette with the data base(s) on it to:

THE ALTERNATE SOURCE
1806 Ada Street
Lansing, MI 48910

TAS will review the adventure for originality and general bugs. The adventure may not be acceptable because it is not original (a copy of someone else's) or is thought not to be in good taste. Or, heaven forbid, it just may not be good enough. Before sending in any adventures ask yourself if you would buy it if you saw it?

The diskette may returned with some suggestions for improvement. TAS also reserves the right to make simple changes to the data base to improve its play.

If your adventure is accepted, you will be notified. A contract will be sent to you upon acceptance discussing the royalty payment. You may decide not to market the adventure without going through TAS. If you so choose, remember that the adventure driver program "ADV" is copyrighted and can not be sold with your adventures unless written permission is given by the author.

Please allow at least 4 weeks for the selection procedure.